Carnegie Mellon
Software Engineering Institute

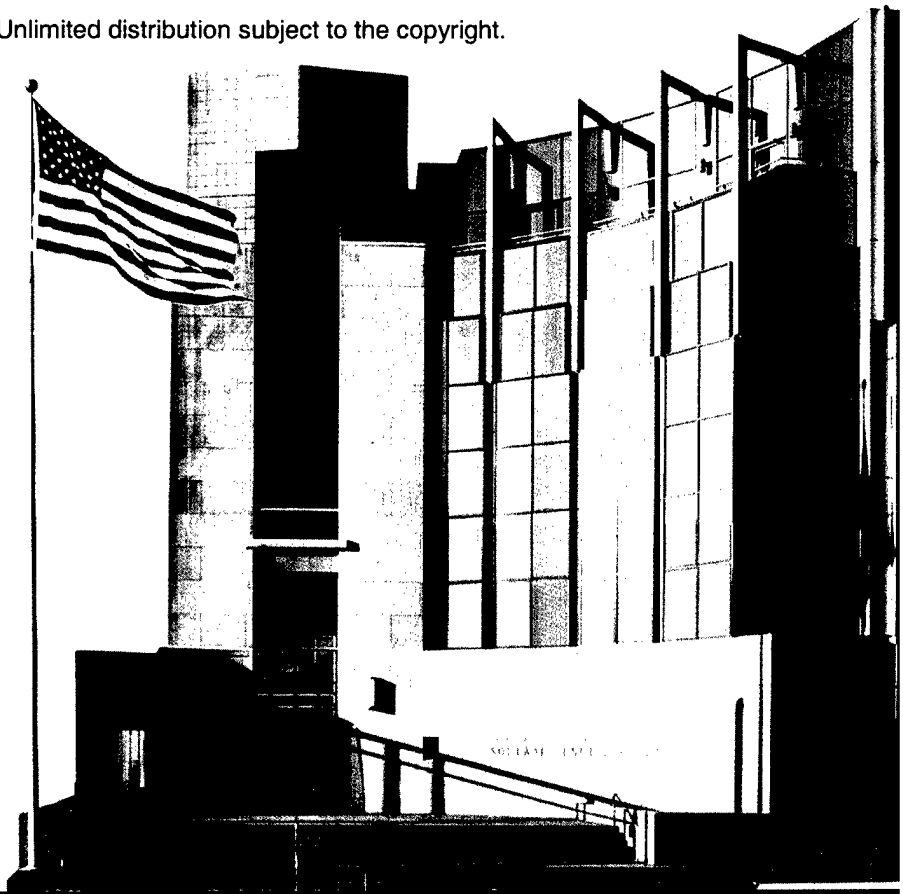
The Software Engineering Institute's Second Workshop on Predictable Assembly: Landscape of Compositional Predictability

Judith A. Stafford
Scott Hissam

June 2003

Predictable Assembly from Certifiable Components Initiative

Technical Note
CMU/SEI-2003-TN-029



The Software Engineering Institute's Second Workshop on Predictable Assembly: Landscape of Compositional Predictability

Judith A. Stafford
Scott Hissam

June 2003

Predictable Assembly from Certifiable Components Initiative

Technical Note
CMU/SEI-2003-TN-029

Contents

List of Tables

Acknowledgements

Our sincere appreciation goes to the workshop participants who shared their knowledge and experiences willingly.

Abstract

To further its work in predictable assembly focusing on compositional reasoning techniques, the Software Engineering Institute (SEI[SM]) held its second Predictable Assembly from Certifiable Components (PACC) Workshop on January 10-11, 2003. Six leading researchers in component-based software engineering were invited to discuss topics related to compositional reasoning with members of the SEI technical staff. During the workshop, participants articulated the current state of research, identified gaps in the available technology, and set the direction for future efforts.

1 Introduction

1.1 Why Predictable Assembly?

Across the software industry, projects face the challenge of satisfying ubiquitous demands for increased functionality, better quality, and reduced cycle time (time to market or field), while simultaneously ensuring that delivered software and software systems satisfy security, survivability, availability, and interoperability requirements. While the use of component-based development (CBD) promises to address the first set of demands, current processes and technologies fail to help developers predict the qualities of a system of components, resulting in expensive integration and testing efforts. Exploiting the full potential of CBD to meet these demands requires the software industry to develop improved, enhanced, or new processes, methods, and tools for determining the properties of software systems before they are built and for confirming their "as-built" properties.

The software industry has developed numerous technologies—such as .NET, Enterprise JavaBeans (EJBs), and the Common Object Request Broker Architecture (CORBA)—to assemble systems from components that are created in isolation. Significant economic and technical benefits from component approaches have accrued. However, component approaches miss the mark in being able to substantively address some of the real issues, namely, the inability to predict individual component behavior, the behavior of assembled components, and hence, the quality of the system. Component technologies available today allow system builders to plug components together, but do little to allow the builder to ensure how well they will play together. As a result, there are rampant failures with component assemblies that receive inadequate testing, and expensive integration and test cycles on those that are made to succeed. Consequently, it is not surprising that there is a lack of consumer trust in the quality of software components and in the quality of assemblies that have not received extensive and expensive testing.

In short, though the component approach has helped and does hold promise, it does not now address the real challenges; software components are critical to the software industry, but the behavior of component assemblies is unpredictable. The impetus for today's component technology was inarguably the needs of the exponentially growing information technology (IT) industry. However, software component technology has not yet demonstrated an ability to predictably meet the requirements for scale, robustness, and performance that have IT-bound organizations. The results are unpredictable and costly development, decreased assurance of how the delivered system will behave, and ultimately, slowed adoption of component technology.

The fundamental technical reasons behind these inadequacies are

- Component interfaces are not sufficiently descriptive.
- The behavior of components is, in part, an *a priori* unknown.
- Consequently, the behavior of component assemblies must be discovered.

To compound these nontrivial technical reasons, there is, in general, a lack of consistency in how the software industry speaks about components and many misunderstandings as to what a component really is, what current component technologies provide, and where commercial off-the-shelf (COTS) components fit into the component discussion. Some people equate components exclusively with COTS components, while others equate component technologies with a specific vendor's offering; for example, EJBs. Of course, such inconsistencies and misunderstandings lead people to make poor decisions.

To achieve the promised potential of component technology and make that technology effective for the software industry, there must be focused technical leadership to crystallize component concepts and channel researchers to address their current inadequacies; to find ways to predict the behavior of an assembly of components before the components are developed, purchased, installed, and integrated; and, finally, to improve the level of trust that can be associated with software components.

1.2 What Is PACC?

The Software Engineering Institute (SEI[SM]) is assuming a leadership role through the newly created Predictable Assembly from Certifiable Components (PACC) Initiative. PACC promises to provide seminal technology for certifying software components in support of predictable assembly. This research—coupled with the SEI's experience with software architecture, component technology, U.S. Department of Defense (DoD) systems, and industrial systems—is unique.

Based on its close connections to and expressed needs from the DoD and industry, PACC assumes that software analyzability, predictability, and certifiability are important and required in increasingly complex mission- and safety-critical systems. Therefore, component technology needs to be infused with a means to not only deliver specific system qualities but also to support design, analysis, prediction, and component assembly.

There are many social and technical barriers to achieving predictable assembly. While the software industry has significant experience with program-level specification, far less is known about system-level specification—in particular, the specification and analysis of

[SM] SEI is a service mark of Carnegie Mellon University.

system-level quality attributes such as security, availability, and fault tolerance. Making progress on this front requires that PACC identifies

- system-level quality attributes of interest to system developers and users
- analysis techniques for reasoning about these quality attributes in terms of components and their patterns of interaction
- design constraints that must be satisfied for the analysis to be valid

Ultimately, all system-level quality attributes depend on the properties of the components that compose the system. But the link between system-level analysis and component properties is very weak and must be strengthened to correlate component properties with system quality. This strengthening requires a more rigorous basis for

- measuring the component properties required by quality-attribute-analysis techniques
- using these properties, in conjunction with reasoning models, to predict system-level quality attributes

In the long run, the challenge is to provide a means of establishing a social and technical basis for trusting components and the engineering predictions based on them.

To surmount these barriers, PACC builds on extensive work both at the SEI and elsewhere in software architecture, quality attribute analysis, component technology, compositional reasoning, and measurement. Based on seminal work at the SEI in software architecture, system quality attributes, and methods for analyzing and designing architectures to support desired system qualities and behavior, PACC relies on the premise that it is possible to restrict software designs in ways that make them analyzable with respect to one or more quality attributes.

Components are then certified to support predictability. This "informative" approach to certification contrasts with, and complements, the more traditional normative approach, where components are certified to satisfy a specific norm. So, for example, PACC would be interested in certifying the latency property of a component, whatever that value is, rather than in certifying that the component satisfies a particular latency norm.

PACC understands that the economic value of assembly prediction and component certification is directly proportional to the accuracy of the predictions and to the time the analysis and prediction effort entails. Therefore, only predictions that are empirically or formally validated are relevant. PACC methods must be mathematically sound and practical.

PACC seeks to augment component technologies with practical, mathematically sound analysis and prediction technologies, so that the quality of systems built from components can be predicted by measuring and certifying component properties. In essence, PACC includes the packaging of design rules so that assemblies of components are by design and

construction analyzable with respect to specific system qualities. These rules will ensure that third parties can certify critical component properties. Certified component properties can then be used to predict the runtime behavior of component assemblies.

1.3 About the Workshop

The PACC Initiative hosted the second PACC workshop, titled "Predictable Assembly: Landscape of Compositional Predictability," on January 10 and 11, 2003 at the SEI in Pittsburgh. This workshop was a part of PACC's ongoing work in nurturing a community of researchers in predictable assembly. The workshop was designed to achieve the following goals:

- Survey the landscape of research in compositional reasoning that supports prediction of the qualities of assemblies of software components.
- Share experiences in the application of compositional reasoning.
- Identify gaps in the practice and research of compositional reasoning.

The workshop follows up on the success of the Fifth International Conference on Software Engineering (ICSE) Workshop on Component-Based Software Engineering (CBSE).[1]

The participants in this workshop were invited based on their experience with component-based software engineering, their published efforts in compositional reasoning, and their expressed interest in predictable assembly.

Participants were encouraged to summarize and present research involving compositional reasoning techniques for application to component assemblies. During their presentations, participants described their problems/research terrains, and focused on the difficulties encountered or lessons learned during their research.

The workshop participants included

- Dirk Beyer, Brandenburg Technical University at Cottbus, Germany
- Ivica Crnkovic, Mälardalen University, Sweden
- Kathi Fisler, Worcester Polytechnic Institute, USA
- Scott Hissam, SEI
- James Ivers, SEI
- Dave Mason, Ryerson University, Canada
- Paulo Merson, SEI

[1] For more information, go to <http://www.sei.cmu.edu/pacc/CBSE5>.

- Gabriel Moreno, SEI
- Heinz Schmidt, Monash University, Australia
- Natasha Sharygina, SEI
- Judith Stafford, SEI/Tufts University
- Rob van Ommering, Philips Research, The Netherlands
- Kurt Wallnau, SEI

1.4 About This Report

This report summarizes the presentations and discussions at the workshop. Section 2 summarizes the presentations given by workshop participants. Section 3 synthesizes the key points of the working session, which covered discussion of both the research and practice in compositional reasoning. The results from the workshop are given in Section 4, which identifies many of the open questions and issues standing as a charge for further exploration to researchers in the software engineering community. Conclusions are in Section 5.

2 Summary of Workshop Presentations

Much discussion took place during the presentations summarized below, permitting participants to grasp a good understanding of each other's research. The issues raised during these conversations provided a basis for discussion during the working sessions on the second day of the workshop. (See Section 3.)

Kurt Wallnau opened the workshop with a presentation of the PACC Initiative's approach to predictable assembly—prediction-enabled component technology (PECT). A PECT is a development infrastructure that guarantees that critical runtime properties of assemblies of components are objectively analyzable and predictable. A PECT comprises a component technology and one or more analysis technologies. Composition tools ensure that component assemblies satisfy analytic assumptions, thus ensuring that assemblies are predictable by construction. Such constructive and analytic assumptions are considered constraints. Predictions about assemblies are manifested through *interpretation*. That is, a constructive assembly is interpreted from its constructive form (e.g., the topology and sequences of component interactions) into a form that is suitable for analysis and prediction (e.g., an analyzable model for performance). The two fundamental ideas behind a PECT are as follows:

1. A component technology imposes constructive and analytic constraints, and provides tools and environments that enforce them. As a result, component assemblies have predictable behavior by construction—hence the term *predictable assembly*.
2. Interpretations are defined to transform component and assembly specifications into analyzable models of assembly behavior. As a result, component properties required for predictability are defined unambiguously, establishing a basis for trust and certification—hence the term *certifiable components*.

Kurt went on to differentiate the PECT approach in supporting reasoning about composition from the topic of compositional reasoning. According to de Roever, compositional reasoning is a precise formulation of reasoning about entire assemblies by considering the pair-wise composition of any two components (or modules) in that assembly [de Roever 98]. Compositional reasoning, then, is the aggregation of those pair-wise models into a model which is representative of the entire assembly. For PECT, compositional reasoning can be too strong in that not all analysis models can independently reason about the pair-wise composition of components in an assembly in isolation. This fact was illustrated in an example that dealt with the latency of an assembly of software components. The assembly's timing properties could not be decomposed into independent subproperties because the timing behavior of one component might influence that of others—and not just the specific

component it's paired with (or connected to). In that context, PECT does not always support compositional reasoning. It does, however, always support reasoning about compositions.

Rob van Ommering presented the work that his team is doing at Philips in predicting properties of assemblies of Koala components. Koala is a component model and an architectural description language that is used to develop software for families of consumer products [van Ommering 02]. Koala components are implemented in C, but communicate with each other solely through interfaces. A Koala interface is a (small) set of functions, parameters, constants, and types, described in an interface definition language (IDL). Components can be assembled into new components recursively, and thus products can be created. The Koala compiler evaluates component bindings as much as possible at compile time and generates code for the rest of the bindings to be resolved at runtime. Koala, which is used to create the software for dozens of products per year, is currently being used by over 100 software engineers. The organization's component base includes over 1,000 components, with a total of several millions lines of code.

Predicting the properties of assemblies of components is an important issue to Philips because the speed of creating products is of the utmost importance in gaining a sufficient market share. The current state of practice at Philips is to measure properties of assemblies after they are compiled, linked, and run. The company thinks that property prediction would give it a significant market edge due to the ability to create new products more quickly. Rob described the prediction of three different types of properties: resource usage (more specifically, the static allocation of resources such as real-time kernel tasks and semaphores), code size (which is an important issue for Philips because it typically works with resource-constrained products), and timing properties associated with multi-threaded environments. Philips is currently considering the calculation of other properties of assemblies as well.

Kathi Fisler presented the work that she is doing at the Worcester Polytechnic Institute (WPI) in developing a compositional reasoning framework that supports the application of verification tools when the feature-oriented programming approach to system development is used. Feature-oriented programming organizes programs around features rather than objects, which is claimed to better support extensible, product line architectures. Programming languages increasingly support this style of programming, but programmers using the style get little support from verification tools. Feature-oriented programming uses a different composition semantics than that of modern verification tools. As a result, feature-oriented programming requires new theories of compositional reasoning. Kathi is developing a compositional reasoning framework in which programmers can verify features independently of each other. Programmers can use automated techniques, such as model checking or automated compositional reasoning techniques, to infer properties of a whole system from those of its features. The framework supports automated derivation of sufficient interfaces on feature components to enable this compositional reasoning.

To build this framework, she has confronted interesting issues in reasoning about open systems and reasoning with information whose definition evolves as new features are added to systems. Kathi's presentation surveyed these broader issues, discussed how the use of feature-oriented programming affects the notions of component composition, and presented the WPI's approach to feature-oriented compositional verification.

Ivica Crnkovic talked about emerging properties for component-based safety-critical systems based on his work at SAVE.[2] SAVE is concerned about conflicting requirements in distributed real-time vehicular systems. His presentation focused on extra-functional properties for safety-critical systems, in particular vehicular systems. For those types of systems (e.g., cars, train systems, and aircraft), the component-based approach to development has a relatively long tradition, as these types of systems are typically built from physical components. Today, the physical components also include several computer nodes (or electronic control units [ECUs]) equipped with software that implements vehicle functions. Current work is focused on moving from a "one node/one supplier" development paradigm to a situation with "one node/several suppliers," that is, a paradigm where several software components of different origins execute on a particular node. Satisfactory handling of safety-critical functions, such as emergency "brake and steer by wire" systems, requires the integration of methods for establishing functional and temporal correctness for each component, as well as system properties such as safety and reliability. Satisfactory properties must be established while minimizing resource demands and maximizing reuse to keep costs and the time to market competitive.

For safety-critical systems (i.e., those systems whose failure can cause human casualties), extra-functional properties are of primary importance. In most cases, safety-critical systems are also hard-real-time systems (i.e., systems in which the timing requirements must be met) and embedded systems (i.e., combinations of software and hardware), which implies the existence of a number of timing and resource constraints. Such systems require rigorous development procedures to make system behavior as predictable as possible. The CBD approach is as attractive in this domain as for other domains, but it is less feasible because it does not provide accurate solutions for extra-functional requirements.

Ivica raised interesting questions related to dependability properties, including which of these properties are emergent system properties (i.e., the system properties do not exist on the component level). Which are both system and component properties? How are these properties in a component-based system related to component properties? To what extent (and how) can these properties be determined from component properties? To what extent can uncertainty in the predictability of these properties be minimized, and how much is that related to the uncertainty of the component properties? Is there a particular phase of development in which these properties are best evaluated? In addition to discussing these questions, Ivica presented a list of additional extra-functional properties classified according

[2] For more information, go to <http://www.idt.mdh.se/~mhn/save/>.

to the degree that they can be understood based on combining component properties or the degree of attention they require to the context in which the component is used.

Dirk Beyer described a model-checking tool called Rabbit[3] that is used to evaluate assemblies described using Cottbus Timed Automata—a modular modeling language based on timed automata. The current version of Rabbit provides verification based on binary decision diagrams (BDDs) using an integer semantics. Dirk described the use of Rabbit for reachability analysis as well as refinement checking to enable the application of compositional reasoning. Rabbit uses the component structure of the model to determine good variable orderings that will reduce the size of the resulting BDDs. Dirk described various case studies that demonstrate the practicability of the approach.

Heinz Schmidt presented the work that his group is doing studying ideas for a partially compositional approach to predicting properties of assemblies of components. He also discussed problems associated with compositionality for extra-functional reasoning models and the software architect's concern with both functional and extra-functional design. He noted that in component-based software engineering, an important task in functional design is the adaptation of a component interface for use by other components, and, in extra-functional analysis, the focus is on the prediction and reasoning about performance, reliability, usability and other *'ilities*. These 'ilities are often system properties that depend on other components in the environment, and the architecture or framework in which the component is deployed.

Heinz presented a concurrent trace-based model and method for component composition and automatic adaptation called Dependent Finite State Machines (DFSMs). DFSMs are based on finite state machines and Petri nets, and permit model checking and execution-based verification. They also extend the notion of "design by contract" from pre-condition, post-condition, and invariant assertions on objects to dynamic models for components that cater to parameters actualized at deployment time. While DFSMs capture only limited aspects of component behavior, they are capable of carrying various extra-functional properties as add-on attributes.

Dave Mason described his work on creating a context-sensitive theory for identifying operational profiles of components. Dave noted that composable program properties fall into two classes: static and probabilistic. Static properties are better known and understood. They can be determined from a structural analysis of the individual components involved in a composition, as well as the interconnections arising from the composition. The nice characteristic of these properties is that they are independent of the inputs to the composition, so a single determination of the property metric is usable for all uses of the particular composition.

Probabilistic properties of a component depend on the input characteristics to the component. Such properties include correctness, runtime, heap-memory utilization, and stack size. For

[3] For more information, go to <http://www-sst.informatik.tu-cottbus.de/~db/Rabbit/1ReadMe.html>.

these properties, an analysis of the components and composition can produce a function that maps various input distributions into the desired metric. Subsequently, for each use of the composition, the function can be applied to the input distribution to produce the metric for the composition in the particular context. In some cases, closed forms for the metric function can be generated. In others, the metric function contains one or more iterations that must be performed until a fixed point is reached. Sometimes conservative approximations to the metric can be developed that will be insensitive to the input distributions. Dave noted that there are components for which no theory currently exists to produce input distributions.

3 Working Sessions

The second day of the workshop began with a review of the main points from the previous day's presentations. These points were summarized into several topics that were prioritized according to the group's level of interest in them for further discussion, and then the working sessions were devoted to those topics.

3.1 Summary of Key Points from the Presentation Session

The workshop participants identified a number of topics, or better questions, which were of global interest to the group relative to compositional reasoning. Many participants said that the questions raised were stimulated by the first day's presentations. Unfortunately, the time allotted for the workshop would not permit each question posed to the group to be answered in any great depth, if at all. However, all participants felt that it was important to capture all the questions so that they could be addressed at a later date, if desired.

Component Properties

- Can we annotate properties of components and assume that they are valid in all contexts?
  - For which properties is this possible?
  - Which properties are local, constant?
  - Which properties are parameterized?
  - Which properties are non-local, depending on the system level?
- Is the set of component properties closed or must it be extensible?
  - Which component properties lead to system properties?
  - Which system properties lead to component properties?
  - Is there a link between those properties and requirements?
- What about taxonomies of system quality attributes?
  - What would one look like?
  - How would such a taxonomy be used?
- Which entities have properties/annotations?
  - components, assemblies, behaviors, functions, services, and so forth?
  - Are there annotatable primitives in the constructive world?
  - At what level in the hierarchy of a taxonomy?

- How do hierarchies (or other organizing principles) relate to inductive concepts in the constructive and analytic worlds?

Features, Components/Connectors, and Aspects

- How do they relate to views?
- Is composition hierarchy built-in or extracted for the purpose of reasoning?
- Are aspects built into the design as composable primitives?
- Are views extracted or imposed analytically, or are they made implicit or explicit by some other means?

Substitutability of Components

- type-theoretic approach to component substitutability (e.g., contra-variance)
  - Type theory provides only one <= relation (i.e., subtypes), but other system properties must be preserved under substitution. Therefore, a set of inequality relations define substitutability. What are those relations?
  - Perhaps the inequality relations are not defined exclusively in terms of the properties of the substitutable entities.
  - Substitution may also have side effects on other properties elsewhere—thus, in effect, changing the substitution inequalities on other components (i.e., changing the types of other components).
  - Is this another example of non-locality?

Specification/Implementation Distinction

- In design, it is ideal to specify values of properties as "ranges" that preserve substitutability.
  - Can such specification prevent implementability because such pieces do not meet the specification?
  - Can the effects of emergence be anticipated in such range specifications? (Emergent properties are those that arise from the assembly.)
  - Do "brittle," discontinuous, or chaotic emergent properties exist that make this impossible?
- How much do we need to know about the internal structure/functionality of an implementation?
- Under what circumstances are implementations versus specifications (or abstractions) of implementations necessary to support analysis/prediction?

Discontinuous and Continuous Property Theories

- In physical engineering, everything is continuous and can be dealt with using tolerances.
  - For which properties can we use similar continuous analogies?
  - Are there decidable ways to find areas of discontinuity where the above analogies fail to hold? And where compositional reasoning would break down?

Human Aspects for Specifying Component Properties

- The general sense is that builders are not likely to be able to specify properties because they lack skills, there is little payoff, and they differ for different consumers. Overarching questions include
  - Is the set of such properties even closed?
  - Is it possible to realize the full set in practice?
  - What guidance should be provided in support of predictability (e.g., annotate components with code size)?
  - Will the developers do it? And what will they gain?

What Should Vanilla Components Look Like?

- Which properties should they have (by default)?
- Can their properties be extended by third parties?

Component Hierarchy and Property Theories

- Koala is recursive but, because there are different kinds of work at different system levels (e.g., basic, subsystem product), the idea of "level-specific impact" may be more general than different theories of computation.
- Do the same property theories apply at different "levels" of construction hierarchy (e.g., latency theory at uniprocessor level, shared memory at n-processor)?
- Can the addition of one component require a change to the computational model for theories (e.g., real-time operating system [RT/OS] or any component responsible for coordinating component interaction such as resource management policies, programmable middleware, or containers)?

Feature- Aspect- or Attribute-Oriented Design

- How are these design approaches related to each other and how does each one support (or detract from) "predictable assembly"?

3.2 Discussion

The above questions were collapsed into six topics. The workshop participants were asked to vote on two topics based on their level of interest in discussing them further. The topics were presented as follows:

1. components and their properties
    a. Can we annotate properties of components and assume that they are valid in all contexts?
    b. Is the set of component properties "closed" or must it be extensible?
    c. Which entities have properties/annotations?
    d. What should "vanilla" components look like?
2. hierarchical/compositional systems and reasoning
    a. How do hierarchies (or other organizing principles) relate to inductive concepts in the constructive and analytic worlds?
    b. How are feature-, aspect-, and attribute-oriented design related?
    c. Do the same property theories apply at different "levels" of construction hierarchy?
3. depth of analysis
    a. Under what circumstances are implementations versus specifications (or abstractions) of implementations necessary to support analysis/prediction?
    b. How much do we need to know about the internal structure of an implementation?
    c. When is abstract versus complete reasoning appropriate, inappropriate, or required?
4. substitutability
    a. type-theoretic/property-theoretic inequality relations
    b. the distinction between specification and implementation
    c. We can achieve predictability without substitutability. However, real substitutability requires predictability, and we need substitutability to evolve software.
5. limits of predictability
    a. What about taxonomies of system quality attributes?
    b. Which taxonomies have computational models?
    c. Which taxonomies are computable?
    d. What must we know to perform (or automate) the reasoning?
    e. Are there decidable ways to find areas of discontinuity where the analogies fail to hold?
6. How do we bring all of this to practice?
    a. human aspects for specifying component properties

The participants' votes are shown in Table 1. Because a great deal of discussion had taken place during the summarization and voting, the time remaining was devoted to the single topic of hierarchy—the number one topic of interest.

Table 1: Votes per Topic

Topic #	# of Votes It Received
1	6
2	7
3	5
4	4
5	5
6	0

Koala provides support for and admits the need for hierarchy when developing component-based systems. The group considered the need for hierarchy. Many participants agreed that a primary goal of CBD is to solve large and complex problems through the use of a "divide and conquer" approach to development, which implies decomposition. The group discussed whether it is possible to decompose those problems without being able to compose them. Some members of the group answered no, explaining that if you can decompose into components, you can always find a way of composing. Others pointed out that it is possible to find compositions without prior decomposition, which is the goal of CBD. Much discussion ensued but no consensus was reached. Other benefits of admitting the need for hierarchy are that it helps make both construction and analysis tractable, different components can be assigned to different owners, and it supports reuse, which allows the sharing of software and analysis results.

4 Workshop Results

Participants acknowledged that they learned about additional aspects of compositional reasoning during the workshop. They all agreed that there is a need to pursue deeper understanding of compositional reasoning and predictable assembly. The group compiled the following list of predictability-related topics that were generated during the workshop and that need the software engineering community's attention. The list also includes more general issues related to the topic of compositional reasoning.

1. limits of predictability
    a. Which properties are actually useful/needed in different domains and in practice?
    b. Which computational models can be used to reason about these properties?
    c. Where does predictability "break?" Can we know the limits of our reasoning models?
2. hierarchy
    a. There is a need for better understanding of the alignment of different hierarchies across/within constructive and analytic worlds.
    b. If we reason across levels of construction hierarchy, must the computational model address that hierarchy?
    c. If we want construction-hierarchy-specific computational models, and we want to reason across construction hierarchy levels, we must have theories that unify the level-specific computational models.
    d. We need a uniform computational model at different levels of hierarchy, if we want a uniform reasoning model over the whole hierarchy.
    e. The above observation does not imply that it is desirable to develop a universal computational model.
    f. We want a model just powerful enough to handle all the subcases that will arise in the constructive world.
    g. We may always want to generalize these models to allow their post hoc unification.
    h. We want to know that conclusions made in one computational model will be valid in some other computational model.
3. property locality
    a. Which properties are local, and stable within an assembly and across any other context?
    b. It would be nice to have a list of properties for which locality is possible, or a list of the characteristics of properties that lead to locality.
    c. Is it true that we can't achieve predictability/compositionality per se, but only within certain boundaries?

4. other issues

   a. We need to be far more concrete and less philosophical; we need a much more concrete discussion, while still producing issues that are interesting and representative.

   b. We need to confirm that compositional prediction is quite relevant to industry.

   c. There are more reasons for pessimism arising on the theory side, and discussions of the day have only increased this sense that theoretical solutions may not be possible given the current technology.

   d. The CBSE community needs to continue reaching out to other communities to understand more about what others are doing (i.e., specifying reasoning models in the architecture itself). Concepts and terminology are still unclear, especially across the boundaries of different disciplines/theories.

   e. The need to use hierarchies to manage complexity in practice was agreed to by all participants, implying the need for analysis that is also hierarchical.

   f. The diversity of approaches was interesting, but leads to a need to identify where commonality exists and to devise a way of benchmarking/comparing/improving approaches.

5 Conclusions

The second SEI workshop on predictable assembly proved to be a rewarding, focused forum for identifying many of the issues central to research in compositional reasoning. The participants' breadth of knowledge and experience contributed to lively and informative discussions. The issues included on the list of topics needing the community's attention reflect both participants' increased awareness of the difficult challenges facing researchers in this area and their increased understanding of where progress can be made. The list of challenge topics is a significant contribution because it focuses the research projects in such a way that will lead to a faster solution to the problem of predictable assembly.

References

[de Roever 98] de Roever, W. P. "The Need for Compositional Proof Systems: A Survey," 1-22. *Proceedings of the International Symposium COMPOS'97* (LNCS 1536). Bad Malente, Germany, September 8-12, 1997. New York, NY: Springer-Verlag, 1998.

[van Ommering 02] van Ommering, R. "The Koala Component Model," Part 5, Ch. 12, 223-236. *Building Reliable Component-Based Software Systems.* Crnkovic, I. & Larsson, M. eds. Boston, MA: Artech House, 2002 (ISBN 1580535585).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	June 2003	Final

4. TITLE AND SUBTITLE
The Software Engineering Institute's Second Workshop on Predictable Assembly: Landscape of Compositional Predictability

5. FUNDING NUMBERS
F19628-00-C-0003

6. AUTHOR(S)
Judith A. Stafford, Scott Hissam

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2003-TN-029

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12B DISTRIBUTION CODE

To further its work in predictable assembly focusing on compositional reasoning techniques, the Software Engineering Institute (SEI[SM]) held its second Predictable Assembly from Certifiable Components (PACC) Workshop on January 10-11, 2003. Six leading researchers in component-based software engineering were invited to discuss topics related to compositional reasoning with members of the SEI technical staff. During the workshop, participants articulated the current state of research, identified gaps in the available technology, and set the direction for future efforts.

14. SUBJECT TERMS
component-based software engineering, predictable assembly

15. NUMBER OF PAGES
30

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL